Структура файла BMP.

Все точечные рисунки должны сохраняться с тремя символами расширения .bmp в установленном формате:

BITMAPFILEHEADER. Заголовок файла. Содержит информацию о типе, размере и макете файла, который содержит изображение.

BITMAPINFOHEADER. Заголовок изображения. Содержит информацию об изображении аппаратно-независимого растрового формата(DIB).
Может также использоваться одна из улучшенных версий: BITMAPV4HEADER, BITMAPV5HEADER. Приложения должны использовать член структуры Size, чтобы выяснить, какая информационная заголовочная структура точечного рисунка используется. Для BITMAPINFOHEADER Size=40.

Дополнительный массив структур **RGBTRIPLE** или **RGBQUAD** (палитра). Если BITMAPINFOHEADER.ColorUsed = 0, то палитра отсутствует.

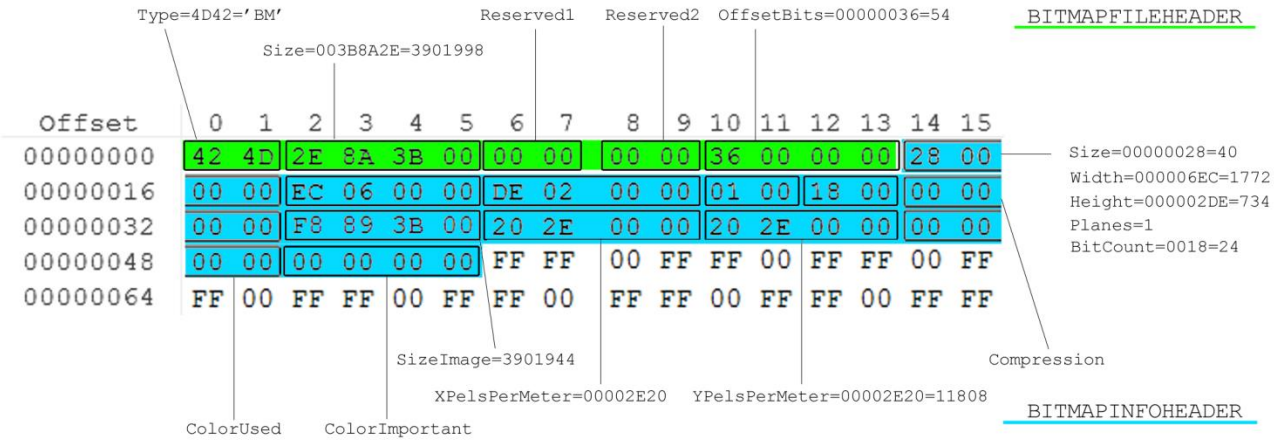
Массив структур **RGBTRIPLE** или **RGBQUAD.** Данные изображения. В файле данные пикселей изображения хранятся построчно, начиная с нижней строки.

```
struct BITMAPFILEHEADER
{
  WORD    Type;      // 'BM' 0x4D42
  DWORD   Size;      // Размер файла в байтах, BitCount*Height*Width+ OffsetBits
  WORD    Reserved1;     // Зарезервирован; должен быть нуль
  WORD    Reserved2;     // Зарезервирован; должен быть нуль
  DWORD   OffsetBits;    // Смещение данных от начала файла в байтах
                         // = sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFOHEADER)
};
```

```
struct BITMAPINFOHEADER
{
  DWORD   Size;            // Число байтов необходимое для структуры = 40
  DWORD   Width;           // Ширина точечного рисунка в пикселях
  DWORD   Height;          // Высота точечного рисунка в пикселях
  WORD    Planes;          // Число плоскостей целевого устройства = 1
  WORD    BitCount;        // Глубина цвета, число бит на точку = 0,1,4,8,16,24,32
  DWORD   Compression;     // Тип сжатия = 0 для несжатого изображения
  DWORD   SizeImage;       // Размер изображения в байтах BitCount*Height*Width
  DWORD   XPelsPerMeter;   // Разрешающая способность по горизонтали
  DWORD   YPelsPerMeter;   // Разрешающая способность по вертикали
  DWORD   ColorUsed;       // Число индексов используемых цветов. Если все цвета = 0
  DWORD   ColorImportant; // Число необходимых цветов = 0
};
```

Все, указанные в комментариях, значения параметров файла и изображения справедливы для 24 и 32 битных беспалитровых несжатых изображений. Изображения с глубиной цвета 1,4,8,16 содержат палитру.

Ниже представлена часть содержимого файла 24 битного беспалитрового несжатого изображения 1772x734 в формате .bmp. Размер файла 3,72 Мб. Для всех пикселей изображения задан бирюзовый цвет(синий и зеленый в равной пропорции).



Данные файла .bmp в шестнадцатеричной записи

Так как палитра для данного изображения отсутствует, то после заголовков следует массив структур RGBTRIPLE. Отличие этой структуры от RGBQUAD заключается в том, что RGBQUAD содержит четвертый байт, который заполняется нулями.

```
struct RGBTRIPLE
{
  BYTE Blue;
  BYTE Green;
  BYTE Red;
};

struct RGBQUAD
{
  BYTE    Blue;
  BYTE    Green;
  BYTE    Red;
  BYTE    Reserved;
};
```

Цвет каждого пиксела в шестнадцатеричной системе счисления задан 3 байтами: FFFF00, то есть интенсивности синий и зеленой составляющей равны FF(255), красной составляющей 00. Для 32 битного изображения этот же цвет будет записан FFFF0000.

В структурах вместо объявленных типов:
BYTE = 1 байт: может быть использован тип данных unsigned char.
WORD = 2 байта: может быть использован тип данных unsigned short.
DWORD = 4 байта: может быть использован тип данных unsigned long.

Создание класса объекта Image. Методы класса для сохранения/создания изображений.

Каждый объект класса Image содержит информацию об изображении, соответствующую структуре BITMAPINFOHEADER и указатель на массив структур RGBTRIPLE или RGBQUAD. Структура BITMAPFILEHEADER в данных объекта не хранится.

```
class Image {

BITMAPINFOHEADER BMInfoHeader;
RGBTRIPLE **Rgbtriple;
// данные могут храниться и в одномерном массиве RGBTRIPLE *Rgbtriple, при этом изменяется способ обращения к пикселу  Rgbtriple[i*Width+j] вместо Rgbtriple[i][j]
RGBQUAD **Rgbquad;


public:

Image (char Mode, unsigned short BCount, int Width, int Height); // Конструктор создания изображения

Image (char *fileName);                    // Конструктор объекта изображения из файла

Image ();       // Конструктор без параметров, создает пустой контейнер под изображение

Image (const Image &i);      // Конструктор копии

~Image ();                   // Деструктор

int loadimage(char *fileName);   // метод загрузки изображения аналогичный конструктору

void writeimage(char *fileName); // метод записи изображения в файл

Image operator = (Image Inp);    // Перегрузка оператора =

// …

};
```

Примеры использования методов класса:

Image img("b", 24, 480, 640); - создание объекта белого 24 битного изображения размером 480*640, остальные параметры заголовка заполняются стандартно или вычисляются

Создание объекта класса Image, содержащего данные из файла input.bmp можно осуществить двумя способами:

```
Image img("input.bmp");
или
Image *img;
img=new Image;
img->loadimage("input.bmp");
```

Конструктор копий вызывается в случае, когда необходимо выделение памяти для копии:

```
Image img2=img;
```

img2.writeimage("output.bmp"); - сохранение изображения в файл

В качестве примера инициализации заголовка, выделения памяти и заполнения в зависимости от параметра mode данных изображения можно рассмотреть метод создания нового изображения (24 или 32 бит), реализованный в виде одного из конструкторов класса Image:

```
Image (char Mode, unsigned short BCount, int Width, int Height)
      {

// Ининциализация заголовка изображения, все параметры задаются стандартного или вычисляются на основании BCount, Width и Height, для палитровых изображений вычисление некоторых переменных может отличаться

            BMInfoHeader.SizeH=40;
            BMInfoHeader.Width=Width;
            ...
            ...
            ...
            BMInfoHeader.ColorUsed=0;

// В данной реализации предусмотрено хранение данных изображения как в RGBTRIPLE, так и в RGBQUAD

            if (BMInfoHeader.BitCount==24)
            {

      // Выделение памяти для двумерного массива размером Height*Width типа RGBTRIPLE

                  Rgbtriple=new RGBTRIPLE*[BMInfoHeader.Height];
                  for (int i=0;i<BMInfoHeader.Height;i++)
                        Rgbtriple[i]=new RGBTRIPLE[BMInfoHeader.Width];

      // Заполнение данных изображения

                  for (int i=0;i<BMInfoHeader.Height;i++)
                        for (int j=0;j<BMInfoHeader.Width;j++)
                        {
                              Rgbtriple[i][j].Red=Mode;
                              Rgbtriple[i][j].Green=Mode;
                              Rgbtriple[i][j].Blue=Mode;
                        }
            }
            if (BMInfoHeader.BitCount==32)
            {
                  Rgbquad=new RGBQUAD*[BMInfoHeader.Height];
                  for (int i=0;i<BMInfoHeader.Height;i++)
                        Rgbquad[i]=new RGBQUAD[BMInfoHeader.Width];

                  for (int i=0;i<BMInfoHeader.Height;i++)
                        for (int j=0;j<BMInfoHeader.Width;j++)
                              {
                                    Rgbquad[i][j].Red=Mode;
                                    Rgbquad[i][j].Green=Mode;
                                    Rgbquad[i][j].Blue=Mode;
                                    Rgbquad[i][j].Reserved=0;
                              }
            }
      }
```

Считывание и запись данных изображения.

Считывание и запись данных из .bmp файла осуществляется при помощи функций fread и fwrite.

```
BMFileHeader BITMAPFILEHEADER;
//…
FILE *f;
f = fopen("input.bmp","rb"); // необходимо открывать бинарный файл
fread(&BMFileHeader,sizeof(BITMAPFILEHEADER),1,f);
//…
fwrite(&BMFileHeader,sizeof(BITMAPFILEHEADER),1,f2);
```

Для того что сместить текущую позицию считывания к началу данных самого изображения (растру) достаточно воспользоваться функцией fseek(FILE *F, long offset, int origin), где offset – смещение, origin – определяет начало отсчета для смещения(может принимать значения: SEEK_SET - от начала файла, SEEK_CUR – от текущей позиции, SEEK_END – с конца файла):

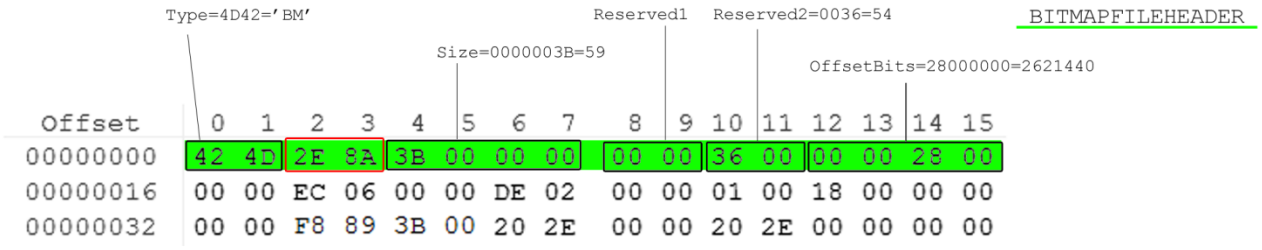
```
fseek(f, BITMAPFILEHEADER.OffsetBits, SEEK_SET);
```

При считывании и сохранении структур могут возникнуть трудности, связанные с тем, что размер структуры по факту может отличаться от суммы размеров всех структурных элементов, так, например:

```
sizeof(BITMAPFILEHEADER)=16
В то время как WORD+DWORD+WORD+WORD+DWORD = short+long+short+short+long =  14
```

Это означает, что данные структуры были выровнены, так как структура BITMAPFILEHEADER состоит из различных типов данных. Это происходит потому, что процессор может обрабатывать только объемы памяти равные машинному слову, и если какой-нибудь и элементов расположен на стыке двух машинных слов, то его удобнее выровнять так, чтобы он располагался от начала второго слова. Аналогичная процедура выполняется для всех элементов структуры компилятором.

Для структуры BITMAPFILEHEADER увеличение на 2 байта происходит из-за выравнивания элемента Size, поэтому при считывании структуры BITMAPFILEHEADER из файла происходит считывание объема данных, не соответствующего установленному.



Данные файла .bmp в шестнадцатеричной записи

В файле заголовок BITMAPFILEHEADER должен занимать 14 байт, поэтому чтобы “укомплектовать” структуру можно воспользоваться дерективой:

```
#pragma pack(push,1)
struct BITMAPFILEHEADER
{
    unsigned short  Type;
    unsigned long   Size;
    unsigned short  Reserved;
    unsigned short  Reserved2;
    unsigned long   OffsetBits;
};
#pragma pack(pop)
```

которая принудительно устанавливает значение выравнивания pack(push,1) для структуры, а затем восстанавливает его ранее значение #pragma pack(pop).

Длины используемых заголовков должны быть:

```
sizeof(BITMAPFILEHEADER)=14
sizeof(BITMAPINFOHEADER)=40
```

Для того чтобы определить тип считываемой структуры данных изображения, используется значение BITMAPINFOHEADER.BitCount, например, если BitCount=24, то память выделяется для массива размером Height*Width типа RGBTRIPLE (аналогично методу: Image (char Mode, unsigned short BCount, int Width, int Height)).

```
RGBTRIPLE **Rgbtriple;
Rgbtriple=new RGBTRIPLE*[BITMAPINFOHEADER.Height];
for (int i=0;i<BMPHeader.Height;i++)
Rgbtriple[i]=new RGBTRIPLE[BITMAPINFOHEADER.Width];
```

Если также создать указатель Rgbstruct:

```
RGBTRIPLE **Rgbstruct=Rgbtriple;
int size=sizeof(RGBTRIPLE);
```

То, так как указатель Rgbstruct ссылается на ту же область памяти, что и указатель массива данных изображения, независимо от того какой тип структуры используется, чтобы считать строку изображения в массив **Rgbtriple, можно использовать:

```
fread(Rgbstruct[i], size, BITMAPINFOHEADER.Width, f);
```

Аналогичная ситуация будет происходить без нужной реализации перегрузки оператора = , то есть по указателю копии imgCopy.Rgbtriple=img.Rgbtriple будет осуществляться доступ к данным оригинала.

Если длина строки не кратна 4 байтам, то после строки следует 1, 2 или 3 нулевых байта.

То есть нужно добавить/считать 4-(BITMAPINFOHEADER.Width*BITMAPINFOHEADER.BitCount/8)%4 байта (00) для того, чтобы начать запись/считывание следующей строки. Дополнение нулями влияет на размер файла (BITMAPFILEHEADER.SizeImage), что необходимо учитывать при определении размера файла.

Задание.

1. Создать класс Image, содержащий информацию об изображении. Все используемые изображения должны быть беспалитровые несжатые. Глубина цвета 24 или 32 бит. Структура, в которой хранятся данные изображения может быть RGBQUAD или RGBTRIPLE в зависимости от варианта.

2. Для хранения используется либо одномерный, либо двумерный массив в зависимости от варианта.

3. Реализовать считывание данных изображения из файла в виде конструктора и метода класса: Image (char *fileName) и int loadimage(char *fileName). Функции должны выполнять следующие операции:
   Считывание и хранение заголовка файла внутри функции
   Считывание и сохранение заголовка изображения в данных объекта. Версии заголовка BITMAPV4HEADER и BITMAPV5HEADER не рассматриваются.
   Выделение памяти под массив данных изображения, на который указывает Rgbtriple,
   Последовательное считывание данных изображения (RGBTRIPLE) или (RGBQUAD), в зависимости от BITMAPINFOHEADER.BitCount. Данные RGBTRIPLE (RGBQUAD) приводятся после считывания к структуре, указанной в варианте.
   В случае ошибки loadimage(char *filename) возвращает 0.

4. Реализовать создание объекта нового изображения конструктором: Image (char Mode, unsigned short BCount, int Width, int Height). Функция выполняет следующее:
   Заполняет структуру BITMAPINFOHEADER в зависимости от входных параметров Width, Height. Поля вычисляются или задаются стандартным образом. Создаваемое изображение беспалитровое несжатое. Глубина цвета 24 или 32 бит.
   Выделяет память под массив данных изображения, на который указывает Rgbtriple(или Rgbquad).
   Записывает данных изображения в зависимости от параметра Mode.

5. Реализовать метод записи изображения в файл void writeimage(char *filename). Функция должна выполнять следующие операции:
   Открытие файла filename для записи.
   Вычисление полей и заполнение структуры BITMAPFILEHEADER. Запись структуры в файл.
   Запись структуры BITMAPINFOHEADER в файл.
   Запись данных изображения в файл построчно, начиная с нижней строки. В зависимости от BITMAPINFOHEADER.BitCount данные преобразуются к нужному типу.
   Закрытие файла

6. Реализовать конструктор без параметров Image (), который не выделяет память под массив данных, все параметры изображения = NULL. Никакие операции кроме присваивания или загрузки изображения из файла с этим объектом выполняться не могут.

7. Реализовать конструктор копий Image (const Image &i) и перегрузку оператора "=" Image operator = (Image Inp), выделяющих память отдельно под копию, это необходимо так как при обычном копировании Rgbtriple(или Rgbquad) копии будет ссылаться на ту же область памяти, что и Rgbtriple(или Rgbquad) оригинала.
   Помимо этого, при перегрузке оператора "=" надо учесть, что присваивание происходит лишь, если левый элемент пуст (это можно определить по BITMAPINFOHEADER.Size=0), в этом случае содержимое объекта копируется полностью. Или изображения имеют один и тот же формат, в этом случае копируются только данные изображения. После того как изображение создано, изменять его параметры (BITMAPINFOHEADER) нельзя.

8. Реализовать деструктор ~Image ().

Масштабирование изображений.

Для масштабирования изображений необходимо добавить оператор в список методов класса:

```
Image operator /= (Image InpImage);
```

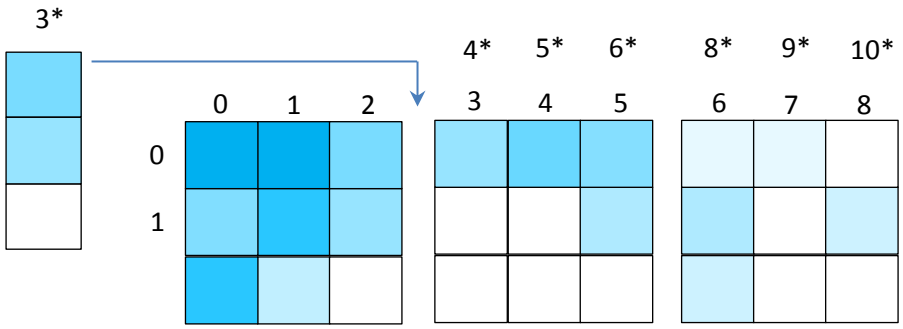
Метод используется для изображений имеющих одинаковую глубину цвета. Пусть изображение input.bmp 24 битное 80*65, тогда после использования оператора /= :

```
Image img("input.bmp");     // Создание объекта изображения из файла
Image img2(0,24,160,65);     // Создание изображения с заданными параметрами
img2/=img;            // Приведение img к масштабу img2
```

В объекте img2 будет храниться такое же изображение, но растянутое по ширине в 2 раза.

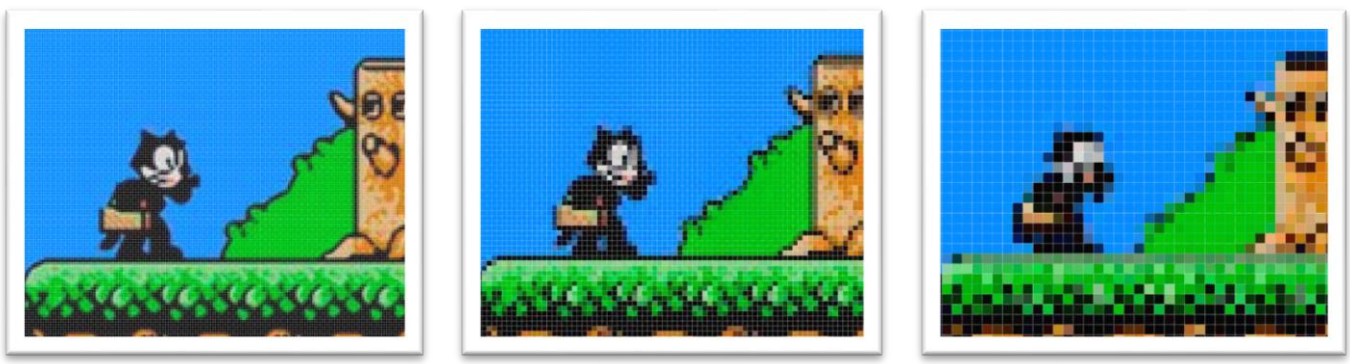
В рассмотренном примере разница по ширине между изображениями составляет 160-80 пикселей, то есть в исходном изображении дана информация о 80 столбцах пикселей и необходимо определить как закрашивать оставшиеся 80 и как эти пиксели располагаются.

Самый простой способ – вставлять(или исключать) в исходное изображение строки и столбцы с определенным шагом так, чтобы его размер соответствовал необходимому. При этом столбцы (строки) дублируют наиближайшие столбцы (строки) исходного изображения.



Увеличение ширины изображения на 25%.
* - соответствует номеру столбца пиксела результирующего изображения

Ниже представлены три изображения: исходное, уменьшенное в 2 раза, уменьшенное в 4 раза. Для иллюстрации уменьшенные изображения были увеличены до размеров исходного.



Пример масштабирования изображений

При добавлении и исключении пикселей могут возникать искажения, несглаженные контуры или пропадать детали. Поэтому вместо того, чтобы просто дублировать (удалять) пиксели, во многих приложениях используются алгоритмы закрашивания пикселей в соответствии с его окружением, такие как Eagle, 2xSal и др.

Палитровые изображения.

Если BITMAPINFOHEADER.BitCount = 1,4 или 8, то после заголовков файла и изображения следует палитра размером $2^{BitCount}$ цветов, которая представляет собой массив структур **RGBTRIPLE** или **RGBQUAD**. Чаще используется **RGBQUAD**, поэтому хранение палитры в **RGBTRIPLE** не рассматривается.

При этом значения BITMAPFILEHEADER.OffsetBits и BITMAPFILEHEADER.Size увеличиваются на размер палитры $2^{BitCount}$*sizeof(**RGBQUAD**), а BITMAPINFOHEADER.ColorUsed присваивается значение $2^{BitCount}$.

Для того чтобы хранить палитру изображения внутри класса Image, необходимо создать указатель Palette.

```
class Image {

BITMAPINFOHEADER BMInfoHeader;
//…
RGBQUAD *Palette;
//…
}
```

Например, изображения в градациях серого представляют собой палитровые изображения, в палитрах которых 256 цветов (8-битное изображение) и для каждого цвета **R=G=B**, то есть палитра имеет следующий вид:

```
00 00 00 00   01 01 01 00   02 02 02 00  .  .  .  FE FE FE 00   FF FF FF 00
```

После палитры 1,4,8-битных изображений следует **не** массив структур **RGBTRIPLE** или **RGBQUAD**, а **индексы цветов в палитре**, каждый из которых занимает BitCount бит. При считывание из файла данных палитрового изображения в массив данных **Rgbquad записывается цвет палитры, на который указывает индекс. Соответственно, при записи данных в файл происходит преобразование цвета из **RGBQUAD** в индекс этого цвета в палитре.

Изменение глубины цвета изображений.

В работе все палитровые изображения считаются изображениями в градациях серого или черно-белыми. Для каждой точки 8-битного изображения значения яркости может быть вычислено:

$$R_8 = G_8 = B_8 = 0.299*R_{24} + 0.597*G_{24} + 0.114*B_{24}.$$

Помимо этого для перевода изображения из 24 или 32 битного формата в палитровый необходимо создавать палитру.

Изменение глубины изображения необходимо реализовать при помощи перегрузки оператора /:

```
Image operator / (short Depth);
```

В общем случае, чтобы привести исходное изображение **img** к определенной глубине цвета необходимо создать пустое изображение (конструктор без параметров) и присвоить пустому

изображению выражение (**img/NewDepth**). При перегрузке важно обратить внимание на то, что оператор / будет использоваться вместе с оператором = :

```
Image img2;    // Создание пустого изображения
img2 =(img/8); // Запись img в img2 с изменением глубины цвета на 8 бит
```

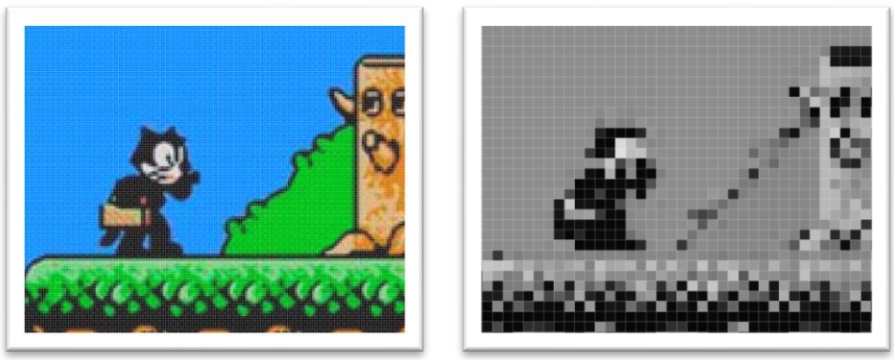
То есть, если img 32-битное изображение, выражение (img/8) будет представлять собой объект класса Image, который содержит изображение, отличающееся от img:

полями BITMAPINFOHEADER.BitCount, BITMAPINFOHEADER.ColorUsed, BITMAPINFOHEADER.SizeImage
выделенной памятью для палитры *Palette
значениями R_8,G_8,B_8 массива данных **Rgbquad

После чего будет происходить присваивание этого объекта объекту **img2**.

Так как оператор /= частично похож на оператор = , возможно следующее совместное использование операторов /= и / :

```
Image img3(0,8,40,32);
img3 /= (img/8);   // img3 содержит изображение img, с установленным размером 40*32 в
                      градациях серого
```



Изображения, хранящиеся в объектах img(размер 160x131) и img3

Изображения с глубиной цвета 1 и 4 бита

Так как при глубине цвета 1 или 4 бита, индекс цвета в палитре занимает часть байта, а на прямую обратиться к области памяти в 4(1) бит невозможно, то необходимо выделять индекс цвета из битового потока.

Рассмотрим содержимое одного байта данных 4 битного изображения (пусть байт хранится в unsigned char Temp):

0	0	1	0	1	0	1	1

В палитре 16 цветов. Старшие 4 бита = 0010, что означает 2 цвет в палитре. Младшие 4 бита = 1011, что означает 11 цвет в палитре. Для того чтобы получить номер цвета в палитре необходимо:

Выделить часть байта. Можно использовать битовую маску, например, mask=0xF0 в двоичном представлении m = 1111 0000, таким образом m&Temp = 0010 0000, что пока равно 32.

Сдвинуть содержимое байта m&Temp на 4 бита вправо. Что даст в результате(m&Temp>>4)=0000 0010.

Для младших битов байта сдвиг не нужен.

Для изображений с глубиной цвета 1 битовая маска будет иметь вид mask=0x80, что в двоичном представлении = 1000 0000. Так как в одном байте 1-битного изображения содержится информация о цвете 8 пикселей, необходимо, сдвигая единицу в маске каждый раз на одну позицию вправо mask>>currbit, где currbit = 0..7, получить индексы всех 8 цветов в байте.

Все 1 и 4 битовые изображения используются в палитре фиксированный набор цветов. Палитра для 1-битных изображений содержит 2 цвета (черный и белый):

00 00 00 00 и FF FF FF 00

Для всех остальных цветов вычисляется:

$$R_8 = G_8 = B_8 = 0.299*R_{24} + 0.597*G_{24} + 0.114*B_{24},$$

после чего из палитры выбирается наиближайший цвет.

Для 4 битных изображений палитра содержит 16 цветов:

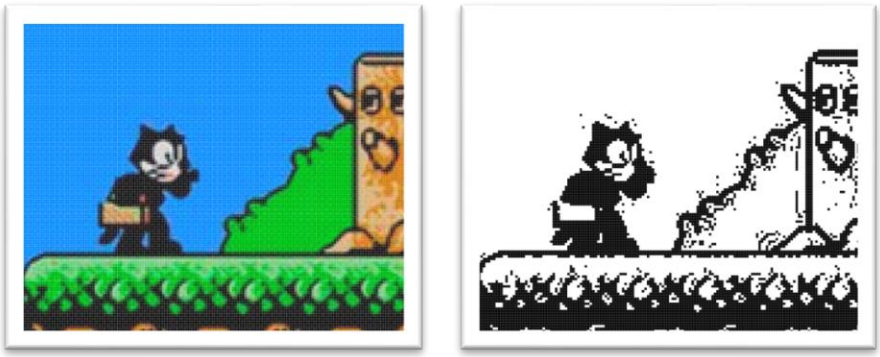
00 00 00 00 0F 0F 0F 00 1F 1F 1F 00 . . . EF EF EF 00 FF FF FF 00

Преобразование 8, 24, 32 битных цветов к 4 битному происходит аналогичным образом.

Если применить преобразование глубины цвета к 1 бит для изображения img:

```
Image img4;
img4 = (img/1);
```

То в объектах img и img4 будут храниться изображения:



Исходное изображение и изображение с глубиной цвета 1 бит

Размер обоих изображений одинаков. Изменение цветов в массиве Rgbquad(Rgbtriple) должно происходить во время вызова оператора /. То есть в img4 цвет каждого пикселя в Rgbquad(Rgbtriple) равен 00 00 00 или FF FF FF.

Задание.

1. Реализовать перегрузку оператора Image operator /= (Image InpImage), осуществляющего масштабирование изображений.

2. Добавить в методы возможность считывания, создания и записи палитровых изображений. Дополняются все методы кроме контсруктора без параметров Image (). Глубина цвета выбирается в зависимости от варианта задания.

3. Реализовать перегрузку оператора Image operator / (short Depth), осуществляющуего изменение глубины цвета.

Варианты задания.

	1	2	3	4	5	6	7	8
1 часть								
Одномерный массив	V		V		V		V	
Двумерный массив		V		V		V		V
RGBTRIPLE	V	V			V	V		
RGBQUAD			V	V			V	V
2 часть								
1 и 8 бит	V	V	V	V				
4 и 8 бит					V	V	V	V

В отчете по курсовой работе необходимо:

Указать номер варианта задания.

Привести описание структуры класса Image, сопоставив её со структурой .bmp файла. Для всех обрабатываемых типов изображений: палитровых и беспалитровых.

Описать все используемые структуры, переменные и методы класса.

Описать и изобразить блок-схему алгоритма масштабирования изображений.

Выбрать 1-2 тестовых изображений.

Продемонстрировать результаты масштабирования и изменения глубины цвета.

Побробно рассмотреть один пример считывания изображения из файла, приведения этого изображения от одного типа (от беспалитрового к палитровому или наоборот) и размера к другому, записи изображения в файл. На каждом шаге необходимо отслеживать размер и содержимое данных всех используемых объектов.

Привести листинг кода программы с комментариями.